Bartosz Reichel

Sieci teleinformatyczne Opis do zadania 1.

Zadanie pierwsze ma na celu zapoznanie się z metodami zapobiegania błędów podczas transmisji. Zapoznamy się tu z trzema podstawowymi metodami:

- Bit parzystości (implementacja dla całego bloku danych).
- Suma modulo (implementacja dla całego bloku danych).
- Cykliczny kod nadmiarowy (CRC).

UWAGA: Niedopuszczalne jest wykorzystanie gotowych funkcji. Szczególnie tyczy się to ostatniego z algorytmów, CRC. CRC należy zaimplementować dla wielomianu o dowolnej długości. Nie wystarczy „pokazanie" ściągniętej z sieci implementacji CRC8, CRC16, CRC32, …

Aby móc porównać te metody, trzeba je zaimplementować. Do porównania będą nam potrzebne pewne wytyczne pozwalające nam zbudować testowy "framework".

Opis aplikacji testowej:

Jako dane testowe będziemy wykorzystywali dowolny plik, należy go wczytać do aplikacji a następnie za symulować transfer danych. W przypadku najprostszym, przekażemy po prostu blok pamięci do funkcji, implementującej jeden z trzech algorytmów (należy zaimplementować wszystkie trzy !). Naturalnie przy takim przesyle (bez błędów transmisji), za każdym razem wynik byłby prawdziwy (tzn. wskazujący na brak błędów). Aby dodać błędy należy zaimplementować funkcje, które dodają błędy. Ilość błędów powinno dać zmienić się (np. 0.1%, 0.01%). Błędy powinny pojawiać się w losowych miejscach (zmianie powinien podlegać 1 bit na jeden błąd). Błędy należy losować na dwa sposoby:

- z powtórzeniami (tzn. ten sam bit może ulec zmianie więcej niż raz),
- bez powtórzeń (jeśli bit raz ulegnie zmianie, nie ulegnie zmianie ponownie).

UWAGA: błąd oznacza zmianę bitu z 1 na 0 lub z 0 na 1, nie liczy się tu zmiana z 1 na 1 lub z 0 na 0 !

Teraz ponownie wyliczamy wartość wedle jednego algorytmu (oczywiście tego samego, z tymi samymi parametrami) za pomocą, którego wyliczyliśmy wynik dla niezaburzonych danych. Porównujemy wyniki ze sobą i zapisujemy (może być ręcznie np. w arkuszu kalkulacyjnym). Porównujemy ze sobą wyniki (przynajmniej 3x5 wyników).

UWAGA: „przesyłając plik" przesyłamy dane z zapisanym wynikiem na końcu naszych danych. Możemy to za symulować zapisując zmieniony plik z wynikiem na końcu. Uproszczony sposób postępowania może być zapisany w postaci punktów:

1) Wczytujemy plik.
2) Wyliczamy wynik na podstawie wybranego algorytmu.
3) Zapisujemy wczytane dane z wynikiem na końcu do pliku.
4) Wczytujemy plik utworzony w pkt 3.
5) Zaburzamy dane (wynik też może ulec zaburzeniu).
6) Wyliczamy ponownie wynik (Znamy format, wiemy że na końcu jest informacja, znamy jej wielkość). Porównujemy z wyekstrahowanym wynikiem z końca pliku. Może się zdarzyć tak, że zniszczeniu ulegnie tylko wynik.

Operacje na bitach

przypominam, że podczas implementacji bardzo pomocne mogą okazać się operacje na bitach takie jak &, |, ^, ~ (notacja z C). Sugeruje przypomnienie sobie podstaw z algebra Boole'a. Dla przykładu podaje tu ustawienie bitu/
Rozważmy bajt 01001 0111, chcemy ustawić w nim 4 bajt (MSB jest na początku LSB jest na końcu)

Bajt: 0100 0111
Tu chcemy mieć „1": 0100 0111

Do tego celu przydatna może być suma logiczna OR (|). Czwarty bajt ustawiony to trzecia potęga dwójki zatem
2^3(DEC) = 8(DEC) = 0000 1000(BIN)
Rozważmy zatem sumę logiczną

```
   0100 0111
OR 0000 1000
------------
   0100 1111
```

Zastanówmy się jak teraz „usunąć" jeden z bitów ?
Tu chcemy mieć „0": 0100 0111.
Powiedzmy niech będzie to bit na drugiej pozycji a zatem związany z pierwszą potęga liczby 2. Z pewnością nie możemy zastosować tu sumy logicznej. Przydatna tu będzie koniunkcja z negacją. Podejście jak poniżej

```
    0100 0111
AND 0000 0010
------------
    0000 0010
```

oczywiście jest niewystarczające (choć przy okazji „odkryliśmy" metodą na sprawdzenie czy bit o numerze n, w tym przypadku n=1, licząc od zera, jest ustawiony). Dokonajmy najpierw negacji liczby 2.

```
NOT 0000 0010
------------
    1111 1101
```

a następnie z tą wartością dokonajmy koniunkcji

```
    0100 0111
AND 1111 1101
------------
    0100 0101
```

wynik gotowy.
Te proste przykłady pokazują trywialność zadania, zachęcam do pracy na kartce papieru aniżeli tracenie czasu na poszukiwanie gotowego rozwiązania w sieci (trochę przypomina to poszukiwania związane z algorytmem dodania dwóch liczb do siebie). Przypominam również o przesunięciach bitowych >> oraz <<, które mogą pomóc w zoptymalizowaniu kodu.